

Sub:	Machine learning and Data analytics using Python					
Date:	04/08/25	Duration:	90 min's	Max Marks:	50	Sem: II

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

1. Use the given Dataset, $x_1=(1,2,2,3,4,5)$ and $x_2=(1,1,3,2,3,5)$ and find the clusters using K-Means clustering algorithm.

Algorithm explanation - 5 marks

Solution - 5 marks

2. Given a dataset, what are the steps in drawing a dendrogram, using Hierarchical clustering (agglomerative method).

Steps – each 2 marks

Diagram – 2 marks

3. Analyze when and why dimensionality reduction techniques such as PCA, LDA should be used before clustering or classification.

PCA

Steps – each 2 marks

LDA

Steps – each 2 marks

4. Explain how t-SNE works for visualizing high-dimensional data in 2D.

T-Sne

Steps – each 2 marks

Diagram – 2 marks

5. Given a small transaction dataset, generate frequent itemsets and extract association rules using the Apriori algorithm.

1-itemset – 2 marks

2-itemset – 2 marks

3-itemset – 2 marks

Evaluation – 4 marks

6. Explain the concept of bagging and boosting in ensemble methods.

Each 5 marks

7. Describe the role of learning rate in Gradient Boosting Machines. How does it impact the model's performance

GBM -5 marks

Definitions – 2 marks

Diagram/Example – 3 marks

8. Analyze the factors that make XGBoost more efficient than traditional GBM.

Explanation – 3 marks

Features – each 2 marks

9. Explain the working of Support Vector Machines. What do you mean by the kernel trick in SVM.

Explanation – 5 marks

Diagram – 3 marks

Kernel trick – 2 marks

10. How does a Recurrent Neural Network (RNN) predicts the next word in a sequence of text data?

Explain.

Concept – 2 marks

Explanation – 3 marks

Layers- 3 marks

Example-2 marks

Solution

PART I

1. Use the given Dataset, $x_1 = (1,2,2,3,4,5)$ and $x_2 = (1,1,3,2,3,5)$ and find the clusters using K-Means clustering algorithm.

K Means Clustering

K means is an iterative clustering algorithm that aims to find local maxima in each iteration. This algorithm works in these 5 steps:

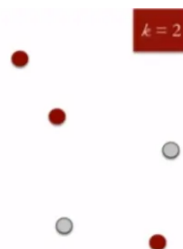
Step1:

Specify the desired number of clusters K: Let us choose $k=2$ for these 5 data points in 2-D space.



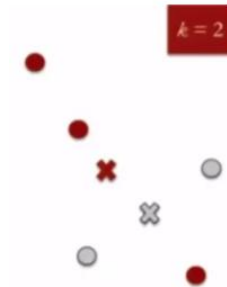
Step 2:

Randomly assign each data point to a cluster: Let's assign three points in cluster 1, shown using red color, and two points in cluster 2, shown using grey color.



Step 3:

Compute cluster centroids: The centroid of data points in the red cluster is shown using the red cross, and those in the grey cluster using a grey cross.



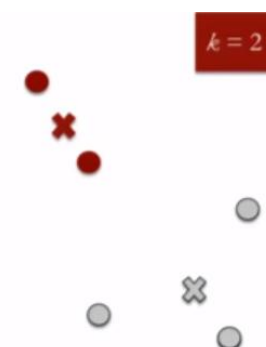
Step 4:

Re-assign each point to the closest cluster centroid: Note that only the data point at the bottom is assigned to the red cluster, even though it's closer to the centroid of the red cluster. Thus, we assign that data point to the grey cluster.



Step 5:

Re-compute cluster centroids: Now, re-computing the centroids for both clusters.



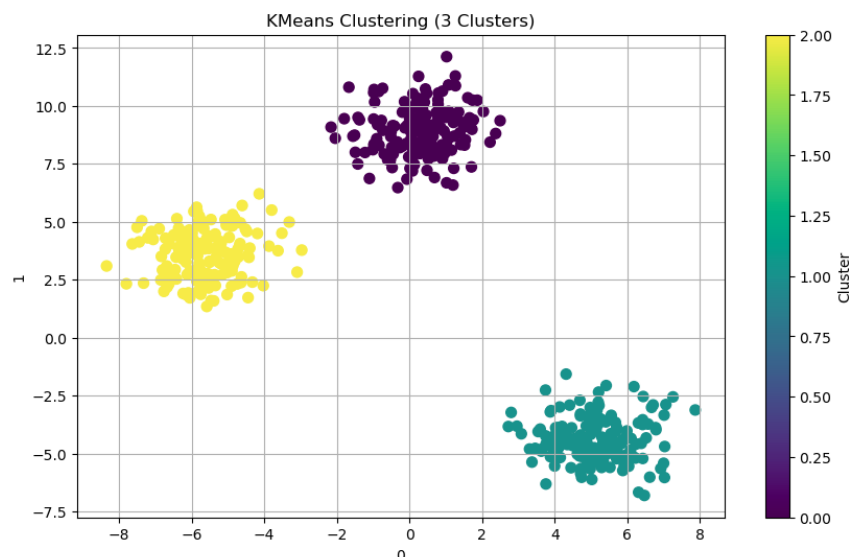
Repeat steps 4 and 5 until no improvements are possible: Similarly, we'll repeat the 4th and 5th steps until we'll reach global optima, i.e., when there is no further switching of data points between two clusters for two successive repeats. It will mark the termination of the algorithm if not explicitly mentioned.

$$d(x, y) = \sqrt{\sum_{i=1}^n (y_i - x_i)^2}$$

1. Choose K number of random data points as initial centroids
2. Repeat till cluster center converges/stabilizes
 - i. Allocate each point in D to the nearest of Kth centroids.
 - ii. Compute centroid for the cluster using all points in the cluster.

		c1=2,1	c2=2,3	Recalculate	c1=2, 1.33	c2=3.67,3.67		Recalculate	c1=2,1.75	c2=4.5,4
x1	x2	Distance from c1	Distance from c2	Assigned center	Distance from c1	Distance from c2	Assigned center	Distance from c1	Distance from c2	Assigned center
1	1	1	2.24	c1	1.05	3.78	c1	1.25	4.61	c1
2	1	0	2	c1	0.33	3.15	c1	0.75	3.9	c1
2	3	2	0	c2	1.67	1.8	c1	1.25	2.69	c1
3	2	1.41	1.41	c1	1.204	1.8	c1	1.03	2.5	c1
4	3	2.83	2	c2	2.605	0.75	c2	2.36	1.12	c2
5	5	5	3.61	c2	4.74	1.88	c2	4.42	1.12	c2

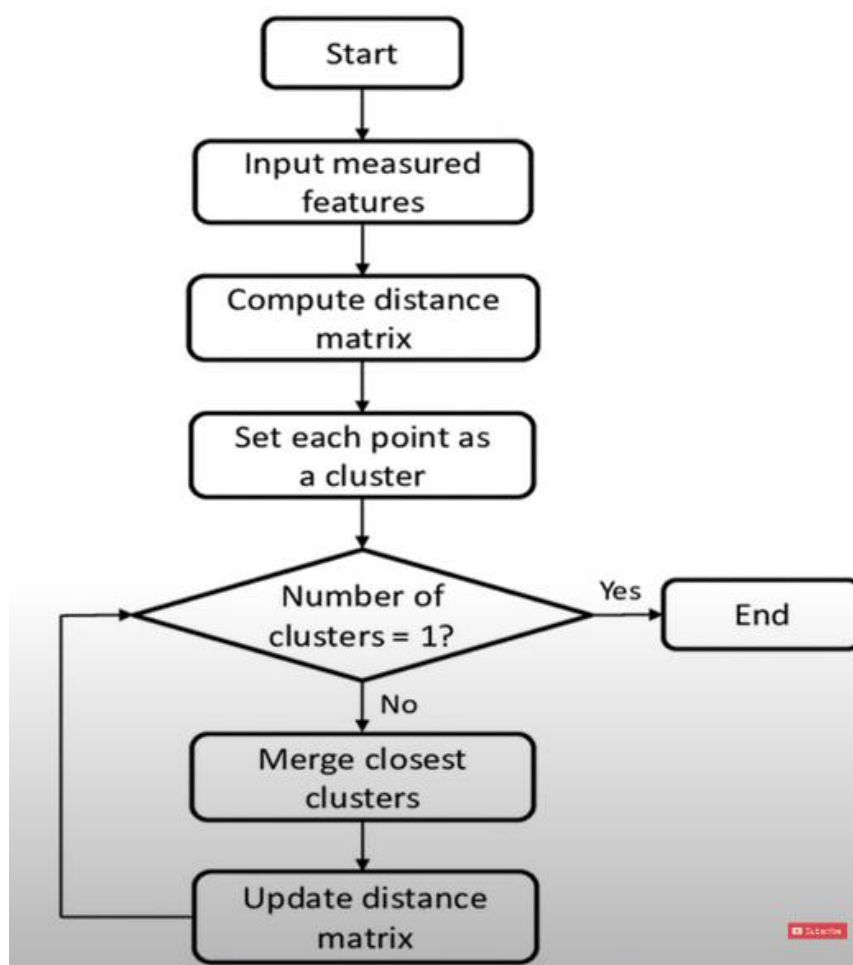
```
# Apply KMeans clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(scaled_data)
```



2. Given a dataset, what are the steps in drawing a dendrogram, using Hierarchical clustering (agglomerative method).

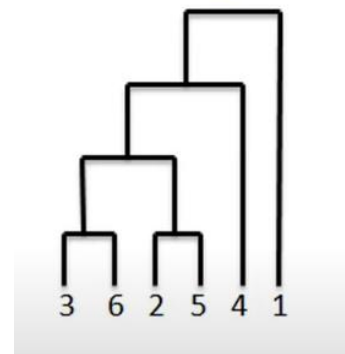
Hierarchical clustering methods, as the name suggests, is an algorithm that builds a hierarchy of clusters. This algorithm starts with all the data points assigned to a cluster of their own. Then two nearest clusters are merged into the same cluster. In the end, this algorithm terminates when there is only a single cluster left. The results of hierarchical clustering can be shown using a dendrogram.

The decision of the no. of clusters that can best depict different groups can be chosen by observing the dendrogram. The best choice of the no. of clusters is the no. of vertical lines in the dendrogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster. In the above example, the best choice of no. of clusters will be 4 as the red horizontal line in the dendrogram below covers the maximum vertical distance AB.



	X	Y
P1	0.40	0.53
P2	0.22	0.38
P3	0.35	0.32
P4	0.26	0.19
P5	0.08	0.41
P6	0.45	0.30

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.23	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

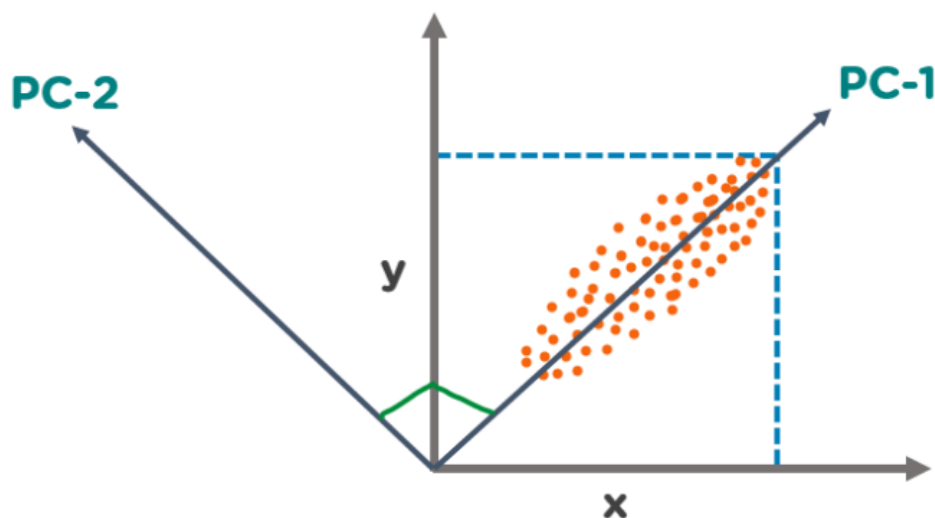


PART II

3. Analyze when and why dimensionality reduction techniques such as PCA, LDA should be used before clustering or classification.

PCA

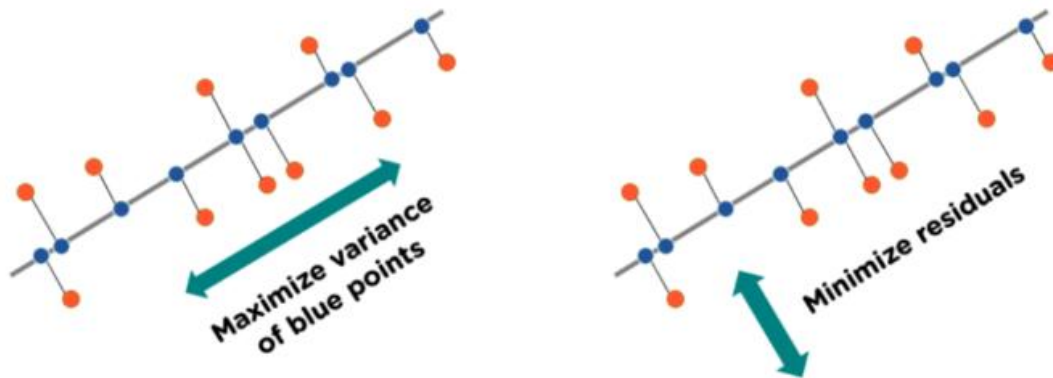
The Principal Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of large data sets. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. PCA helps in finding a sequence of linear combinations of variables.



In the above figure, we have several points plotted on a 2-D plane. There are two principal components. PC1 is the primary principal component that explains the maximum variance in the data. PC2 is another principal component that is orthogonal to PC1.

The Principal Components are a straight line that captures most of the variance of the data. They have a direction and magnitude. Principal components are orthogonal projections

(perpendicular) of data onto lower-dimensional space. The steps involved in PCA are,



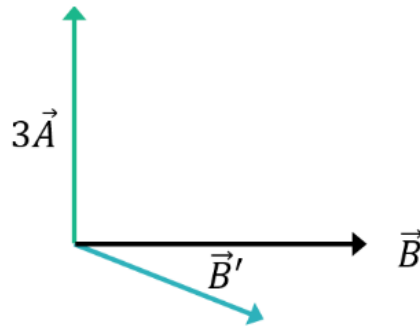
1. Standardize the data: PCA requires standardized data, so the first step is to standardize the data to ensure that all variables have a mean of 0 and a standard deviation of 1.

$$Z = \frac{x - \mu}{\sigma}$$

2. Calculate the covariance matrix: The next step is to calculate the covariance matrix of the standardized data. This matrix shows how each variable is related to every other variable in the dataset.

$$\begin{bmatrix} \text{Var}(x) & \text{Cov}(x,y) & \dots & \text{Cov}(x,m) \\ \text{Cov}(x,y) & \ddots & & \vdots \\ \text{Cov}(z,x) & & \ddots & \vdots \\ \vdots & & & \ddots \\ \text{Cov}(n,x) & \dots & \dots & \text{Var}(n) \end{bmatrix}$$

3. Calculate the eigenvectors and eigenvalues: The eigenvectors and eigenvalues of the covariance matrix are then calculated. The eigenvectors represent the directions in which the data varies the most, while the eigenvalues represent the amount of variation along each eigenvector.



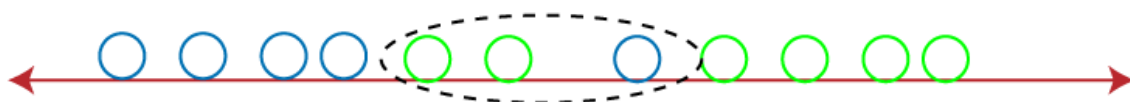
4. Choose the principal components: The principal components are the eigenvectors with the highest eigenvalues. These components represent the directions in which the data varies the most and are used to transform the original data into a lower-dimensional space. Calculate the eigenvectors/unit vectors and eigenvalues. Eigenvalues are scalars by which we multiply the eigenvector of the covariance matrix.
5. Transform the data: The final step is to transform the original data into the lower-dimensional space defined by the principal components.

LDA

Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems.

Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.



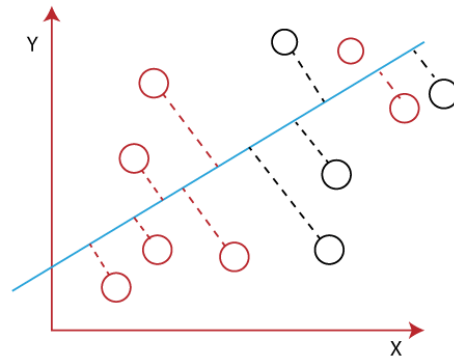
Overlapping

To overcome the overlapping issue in the classification process, we must increase the number of features regularly.

Linear Discriminant analysis is used as a dimensionality reduction technique in machine learning, using which we can easily transform a 2-D and 3-D graph into a 1-dimensional plane.

Let's consider an example where we have two classes in a 2-D plane having an X-Y axis, and we need to classify them efficiently. As we have already seen in the above example that LDA enables us to draw a straight line that can completely separate the two classes of the data points. Here, LDA uses an X-Y axis to create a new axis by separating them using a straight line and projecting data onto a new axis.

Hence, we can maximize the separation between these classes and reduce the 2-D plane into 1-D.



To create a new axis, Linear Discriminant Analysis uses the following criteria:

- It maximizes the distance between means of two classes.
- It minimizes the variance within the individual class.

Using the above two conditions, LDA generates a new axis in such a way that it can maximize the distance between the means of the two classes and minimizes the variation within each class.

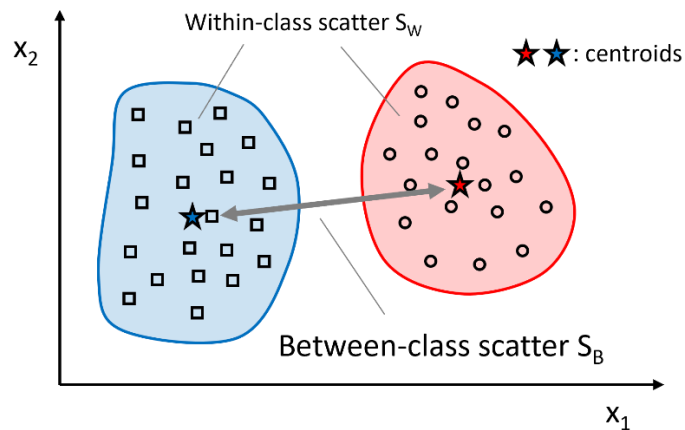
Steps involved in LDA,

Step 1: Compute the d -dimensional mean vectors for each of the k classes separately from the dataset.

Remember that LDA is a supervised machine learning technique, meaning we can utilize the known labels. In the first step, we calculate the mean vectors μ_c for all samples belonging to a specific class c . To do this, we filter the feature matrix by class label and compute the mean for each of the d features. As a result, we obtain k mean vectors (one for each of the k classes), each with a length of d (corresponding to the d features).

Step 2: Compute the scatter matrices (between-class scatter matrix and within-class scatter matrix).

The within-class scatter matrix measures the variation among samples within the same class. To find a subspace with optimal separability, we aim to minimize the values in this matrix. In contrast, the between-class scatter matrix measures the variation between different classes. For optimal separability, we aim to maximize the values in this matrix. Intuitively, within-class scatter looks at how compact each class is, whereas between-class scatter examines how far apart different classes are.



Within-class and between-class scatter matrices

Let's start with the **within-class scatter** matrix S_W . It is calculated as the sum of the scatter matrices S_{c_i} for each individual class:

The **between-class scatter** matrix S_B is derived from the differences between the class means μ_{c_i} and the overall mean of the entire dataset:

where *mean* refers to the mean vector calculated over all samples, regardless of their class labels.

Step 3: Calculate the eigenvectors and eigenvalues for the ratio of S_W and S_B .

The eigenvectors define the directions of this subspace, while the eigenvalues represent the magnitude of the distortion. We will select the m eigenvectors associated with the highest eigenvalues.

Step 4: Sort the eigenvectors in descending order of their corresponding eigenvalues, and select the m eigenvectors with the largest eigenvalues to form a $d \times m$ -dimensional transformation matrix W .

Step 5: Use W to project the samples onto the new subspace.

4. Explain how t-SNE works for visualizing high-dimensional data in 2D.

(t-SNE) t-Distributed Stochastic Neighbor Embedding is a non-linear **dimensionality reduction** algorithm for exploring high-dimensional data. It maps multi-dimensional data to two or more dimensions suitable for human observation. With the help of the t-SNE algorithms, you may have to plot fewer exploratory data analysis plots next time you work with high-dimensional data.

t-SNE, or t-Distributed Stochastic Neighbor Embedding, is a method for converting Visualize high-dimensional data into a more manageable form, typically 2D or 3D, facilitating better comprehension. Here's how it operates:

1. **Measure Similarity:** Initially, it assesses the similarity between each pair of data points within the high-dimensional space.
2. **Map to Lower Dimensions:** Subsequently, it endeavors to represent these similarities in a simplified space while preserving the relationships between the points as faithfully as possible.
3. **Adjust Positions:** It iteratively adjusts the positions of points in the simplified space to align the similarities as closely as feasible.
4. **Visualize:** Finally, it plots these points in the simplified space, often on a graph, facilitating easier pattern recognition and analysis.

Steps in LDA:

- Compute the within-class and between-class scatter matrices.
- Find the eigenvectors and eigenvalues of the scatter matrices.
- Choose the top-k eigenvectors to reduce the dimensionality.
- Project the data onto the linear discriminants.

PART III

5. Given a small transaction dataset, generate frequent itemsets and extract association rules using the Apriori algorithm.

T id	Items
1	Bread, Milk
2	Bread, diaper, Beer, eggs
3	Milk, diaper, Beer, Coke
4	Bread, Milk, diaper, Beer
5	Bread, Milk, diaper, Coke

Consider the following dataset and we will find frequent itemsets and generate association rules for them:

Transaction ID	Items Bought
T1	Bread, Butter, Milk
T2	Bread, Butter
T3	Bread, Milk
T4	Butter, Milk
T5	Bread, Milk

Transactions of a Grocery Shop

Step 1 : Setting the parameters

- **Minimum Support Threshold:** 50% (item must appear in at least 3/5 transactions).

This threshold is formulated from this formula:

$\text{Support}(A) = \frac{\text{Number of transactions containing itemset } A}{\text{Total number of transactions}}$

- **Minimum Confidence Threshold:** 70% (You can change the value of parameters as per the usecase and problem statement). This threshold is formulated from this formula:

$\text{Confidence}(X \rightarrow Y) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}$

Step 2: Find Frequent 1-Itemsets

Lets count how many transactions include each item in the dataset (calculating the frequency of each item).

Item	Support Count	Support %
Bread	4	80%
Butter	3	60%
Milk	4	80%

Frequent 1-Itemsets

All items have **support% $\geq 50\%$** , so they qualify as **frequent 1-itemsets**. if any item has **support% $< 50\%$** , It will be ommited out from the frequent 1- itemsets.

Step 3: Generate Candidate 2-Itemsets

Combine the frequent 1-itemsets into pairs and calculate their support.

For this usecase, we will get 3 item pairs (bread,butter) , (bread,ilk) and (butter,milk) and will calculate the support similiar to step 2

Item Pair	Support Count	Support %
Bread, Butter	3	60%
Bread, Milk	3	60%
Butter, Milk	2	40%

Candidate 2-Itemsets

Frequent 2-itemsets:

- {Bread, Butter}, {Bread, Milk} both meet the 50% threshold but {butter,milk} doesnt meet the threshold, so will be ommited out.

Step 4: Generate Candidate 3-Itemsets

Combine the frequent 2-itemsets into groups of 3 and calculate their support.

for the triplet, we have only got one case i.e {bread,butter,milk} and we will calculate the support.

Item Triplet	Support Count	Support %
Bread, Butter, Milk	2	40%

Candidate 3-Itemsets

Since this **does not meet the 50% threshold**, there are no **frequent 3-itemsets**.

Step 5: Generate Association Rules

Now we generate rules from the frequent itemsets and calculate **confidence**.

Rule 1: If Bread → Butter (if customer buys bread, the customer will buy butter also)

- Support of {Bread, Butter} = 3.
- Support of {Bread} = 4.
- **Confidence** = $3/4 = 75\%$ (Passes threshold).

Rule 2: If Butter → Bread (if customer buys butter, the customer will buy bread also)

- Support of {Bread, Butter} = 3.
- Support of {Butter} = 3.
- **Confidence** = $3/3 = 100\%$ (Passes threshold).

Rule 3: If Bread → Milk (if customer buys bread, the customer will buy milk also)

- Support of {Bread, Milk} = 3.
- Support of {Bread} = 4.
- **Confidence** = $3/4 = 75\%$ (Passes threshold).

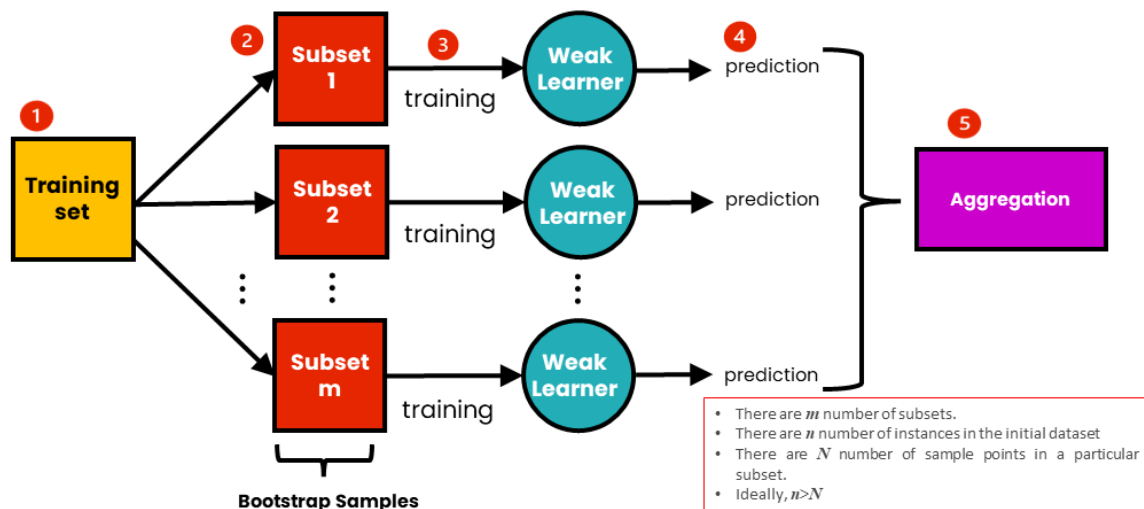
6. Explain the concept of bagging and boosting in ensemble methods.

Bagging (Bootstrap Aggregating) is an ensemble learning technique designed to improve the accuracy and stability of machine learning algorithms. It involves the following steps:

The steps of bagging are as follows:

1. We have an initial training dataset containing n-number of instances.
2. We create a m-number of subsets of data from the training set. We take a subset of N sample points from the initial dataset for each subset. Each subset is taken with replacement. This means that a specific data point can be sampled more than once.
3. For each subset of data, we train the corresponding weak learners independently. These models are homogeneous, meaning that they are of the same type.
4. Each model makes a prediction.
5. Aggregating the predictions into a single prediction. For this, using either max voting or averaging.

The Process of Bagging (Bootstrap Aggregation)



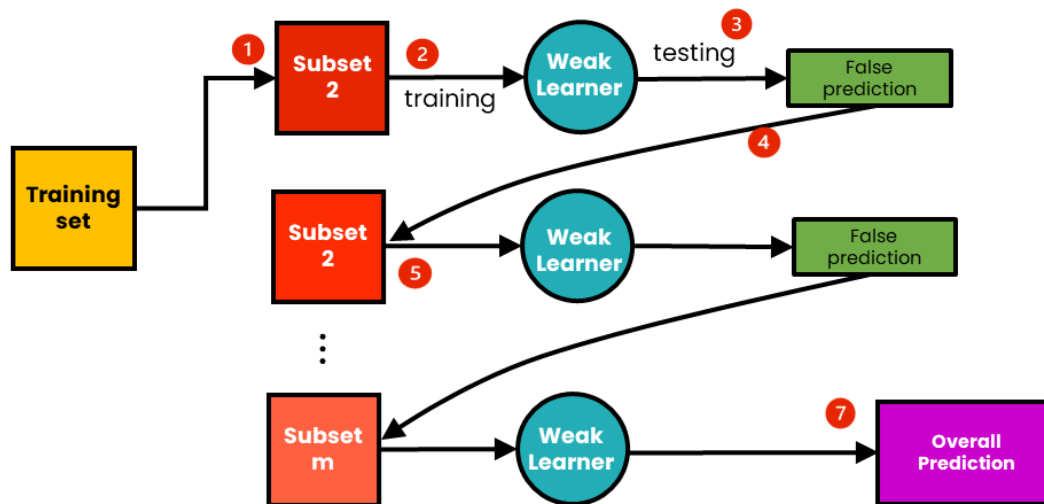
Advantages

- **Reduces Variance:** By averaging multiple predictions, bagging reduces the variance of the model and helps prevent overfitting.
- **Improves Accuracy:** Combining multiple models usually leads to better performance than individual models.
- **Eg. Random Forests** (an extension of bagging applied to decision trees)

Boosting is another ensemble learning technique that focuses on creating a strong model by combining several weak models. It involves the following steps:

- We sample m -number of subsets from an initial training dataset.
- Using the first subset, we train the first weak learner.
- We test the trained weak learner using the training data. As a result of the testing, some data points will be incorrectly predicted.
- Each data point with the wrong prediction is sent into the second subset of data, and this subset is updated.
- Using this updated subset, we train and test the second weak learner.
- We continue with the next subset until reaching the total number of subsets.
- We now have the total prediction. The overall prediction has already been aggregated at each step, so there is no need to calculate it.

The Process of Boosting



Advantages:

- **Reduces Bias:** By focusing on hard-to-classify instances, boosting reduces bias and improves the overall model accuracy.
- **Produces Strong Predictors:** Combining weak learners leads to a strong predictive model.

Example of Boosting Algorithms:

- AdaBoost
- Gradient Boosting Machines (GBM)
- XGBoost

PART IV

7. Describe the role of learning rate in Gradient Boosting Machines. How does it impact the model's performance?

Gradient Boosting:

Gradient Boosting is another popular ensemble learning technique used for regression and classification tasks. It builds a strong predictive model by sequentially adding weak learners to the ensemble, with each new learner correcting the errors made by the previous learner.

Here's how Gradient Boosting works:

1. **Initialization:** Gradient Boosting starts by initializing the ensemble with a simple model such as a simple mean model.

$$F_0 = \text{mean}(y)$$

2. **Residual Calculation:** Now we calculate the residuals (the differences between the predicted values and the actual values) for each training instance. The residuals represent the errors that the current model couldn't capture.

The residuals $r_1 = y - \text{mean}(y)$ is calculated.

3. Weak Learner Training: A new weak learner, typically a decision stump is trained to minimize these residuals. We are building a regression tree model with x as its feature and the residuals $r_1 = y - \text{mean}(y)$ as its target. Please note that gradient boosting trees usually have a little deeper trees such as ones with 8 to 32 terminal nodes. is trained on the residuals from the previous step.

4. This prediction γ_1 is added to our initial prediction F_0 to reduce the residuals. In fact, gradient boosting algorithm does not simply add γ to F as it makes the model overfit to the training data. Instead, γ is scaled down by **learning rate** ν which ranges between 0 and 1, and then added to F .

$$F_1 = F_0 + \nu \cdot \gamma_1$$

5. Now we calculate the residuals r_2 . In the next step, we are creating a regression tree again using the same x as the feature and the updated residuals r_2 as its target.

We iterate these steps until the model prediction stops improving.

Learning Rate: A learning rate hyperparameter is applied to the gradient to control the step size during gradient descent. Lower learning rates lead to slower learning but can improve the model's generalization.

Model Update: The predictions of the weak learners are updated by taking a step in the direction of the negative gradient, scaled by the learning rate. This process is repeated for a predetermined number of iterations or until a certain threshold of performance is reached.


```
# Split dataset into test and train data
X_train, X_test, y_train, y_test = train_test_split(df.drop(' income', axis=1), df[' income'], test_size=0.2)

# for encoding to convert categorical to binary
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)

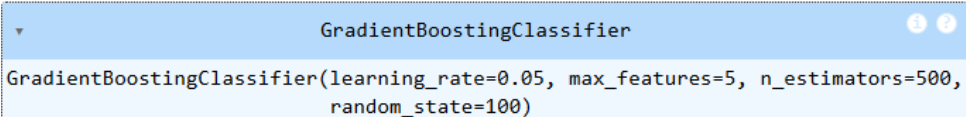
# Ensure both sets have the same columns
X_train, X_test = X_train.align(X_test, join='left', axis=1, fill_value=0)

# Define Gradient Boosting Classifier with hyperparameters

gbc = GradientBoostingClassifier(n_estimators=500, learning_rate=0.05, random_state=100, max_features=5)

# Fit train data to GBC

gbc.fit(X_train, y_train)
```



8. Analyze the factors that make XGBoost more efficient than traditional GBM.

XGBoost, short for eXtreme Gradient Boosting, is an optimized and highly efficient implementation of gradient boosting. It is one of the most popular and widely used machine learning algorithms, particularly in competitions like Kaggle, due to its speed, scalability, and outstanding performance. Gradient boosting machines are generally very slow in implementation because of sequential model training. Hence, they are not very scalable. Thus, XGBoost is focused on computational speed and model performance, while introducing regularization parameters to reduce overfitting.

- **Regularization:** XGBoost regularize complex models through **L1 and L2** regularization techniques - **Regularization helps in preventing from overfitting.**
- **Handling sparse data:** XGBoost Classifier incorporates a **sparsity-aware split** finding algorithm to handle different types of sparsity patterns in the data (missing values) – Eg. one-hot encoding make **data sparse.**
- **Weighted quantile sketch:** XGBoost has a distributed weighted quantile sketch algorithm to effectively **handle weighted data.**
- **Block structure for parallel learning:** For faster computing, XGBoost Classifier can make use of multiple cores on the CPU. This is possible because of a block structure in its system design.
- **Cache awareness:** Tianqi Chen designed XGBoost to optimize hardware usage. This optimization occurs by allocating internal buffers in each thread, where the workflow can store the gradient statistics.
- **Out-of-core computing:** This feature optimizes the available disk space and maximizes its usage when handling huge datasets that do not fit into memory

Feature	XGBoost	Gradient Boosting
Description	Advanced implementation of gradient boosting	Ensemble technique using weak learners
Optimization	Regularized objective function	Error gradient minimization
Efficiency	Highly optimized, efficient	Computationally intensive
Missing Values	Built-in support	Requires preprocessing
Regularization	Built-in L1 and L2	Requires external steps
Feature Importance	Built-in measures	Limited, needs external calculation
Interpretability	Complex, less interpretable	More interpretable models

PART V

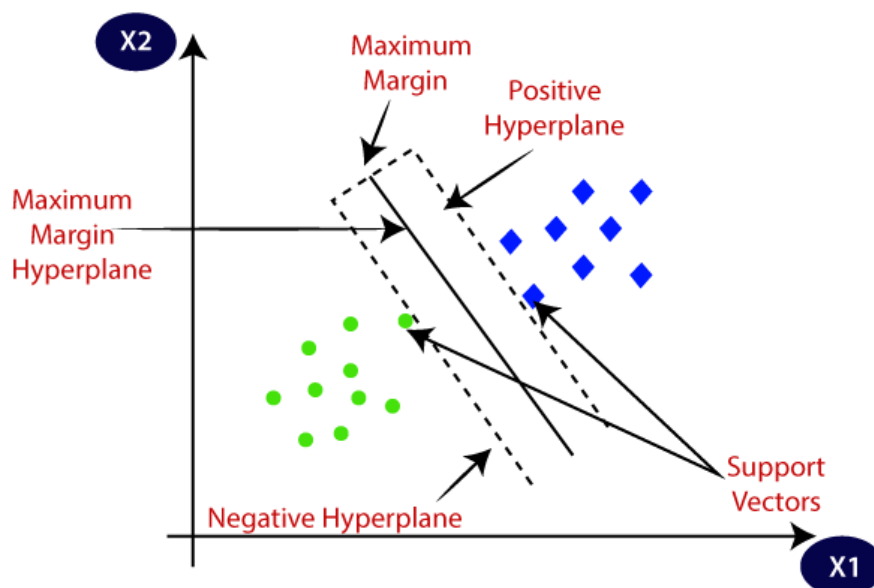
9. Explain the working of Support Vector Machines. What do you mean by the kernel trick in SVM.

Support Vector Machines

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



SVM algorithm can be used for **Face detection, image classification, text categorization**, etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

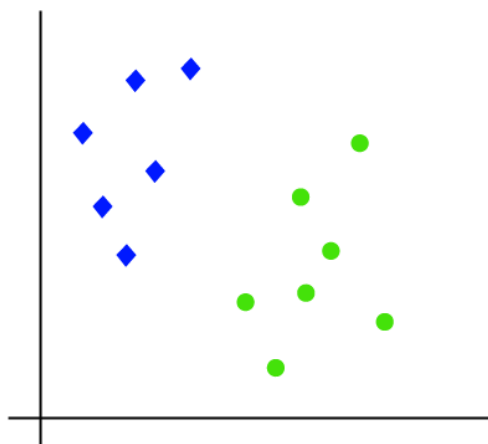
Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

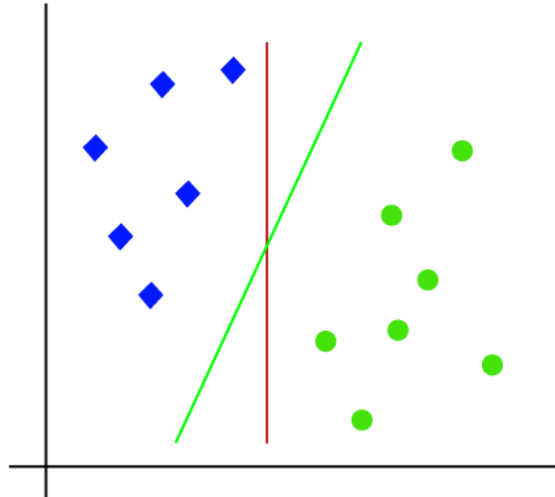
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

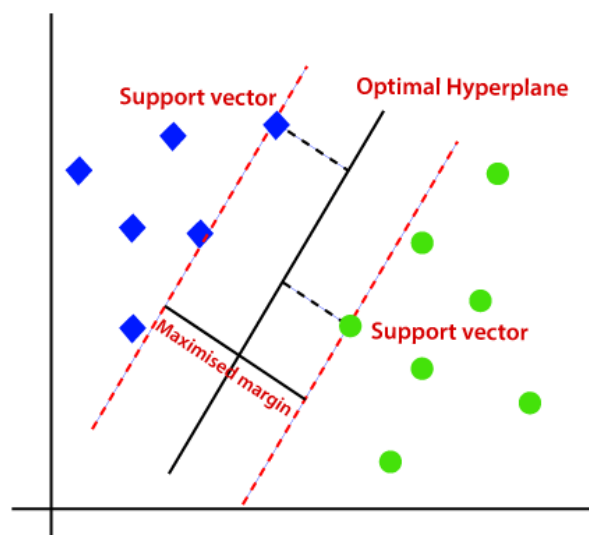
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x_1 and x_2 . We want a classifier that can classify the pair(x_1 , x_2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.

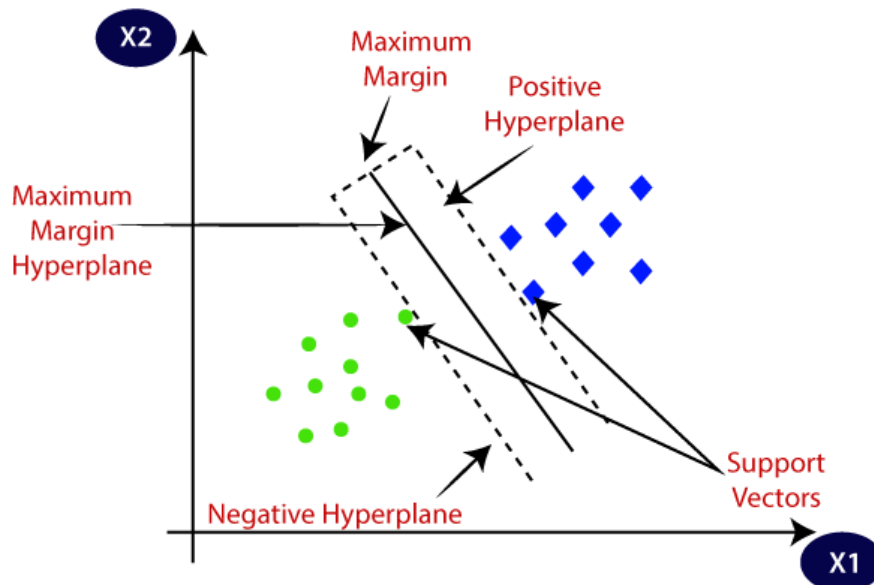


Support Vector Machines

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



SVM algorithm can be used for **Face detection, image classification, text categorization**, etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n -dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

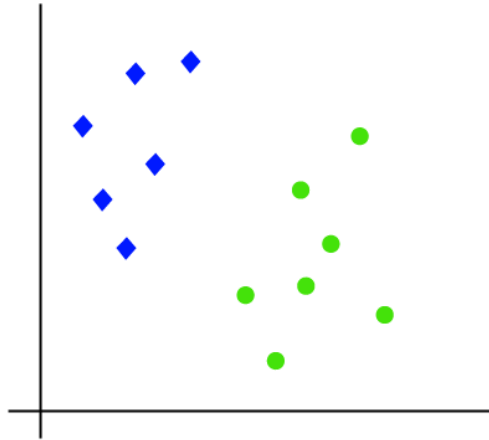
The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Support Vectors:

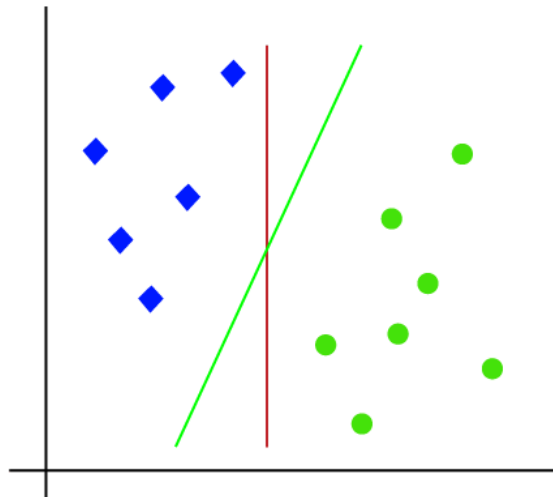
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x_1 and x_2 . We

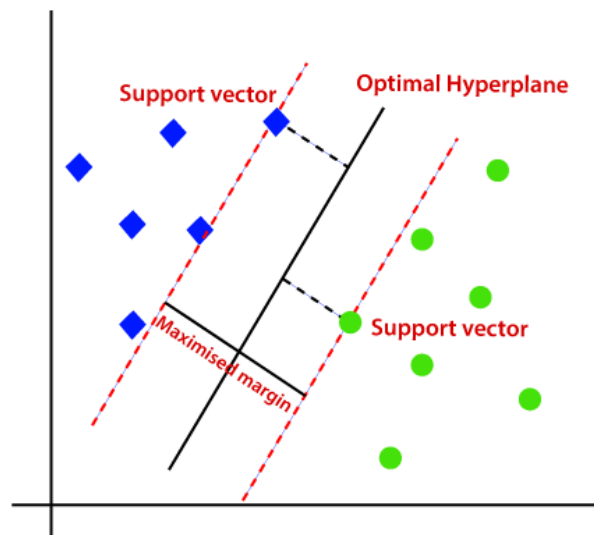
want a classifier that can classify the pair(x_1 , x_2) of coordinates in either green or blue. Consider the below image:



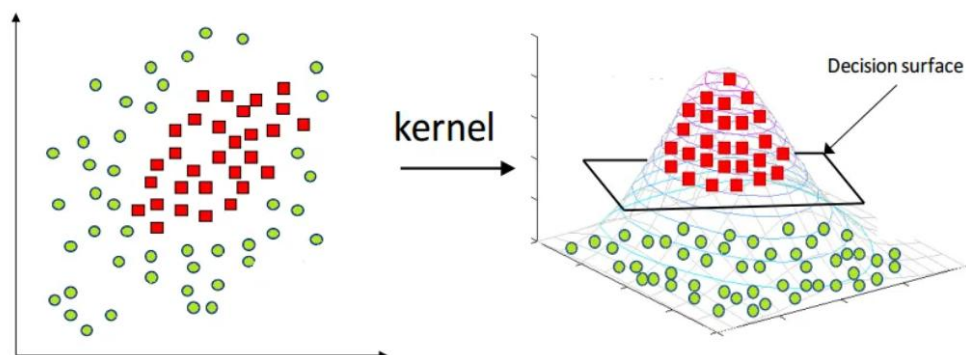
So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



- A kernel is a function that maps data points into a higher-dimensional space **without explicitly computing the coordinates in that space**.
- This allows SVM to work efficiently with non-linear data by implicitly performing the mapping.
- By applying a kernel function SVM **transforms the data points into a higher-dimensional space** where they become linearly separable.

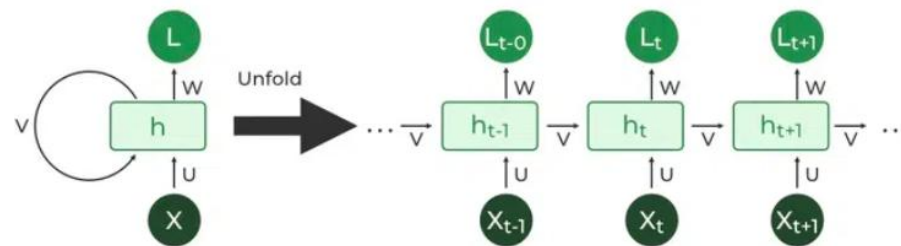


10. How does a Recurrent Neural Network (RNN) predicts the next word in a sequence of text data? Explain.

Recurrent Neural Networks (RNNs) work a bit different from regular neural networks. In neural network the information flows in one direction from input to output. However, in RNN information is fed back into the system after each step. Think of it like reading a sentence, when you're trying to predict the next word you don't just look at the current word but also need to remember the words that came before to make accurate guess. Recurrent Neural Networks (RNNs) solve this **by incorporating loops that allow information from previous steps to be fed back into the network**. This feedback enables RNNs to remember prior inputs making them ideal for tasks where context is important.

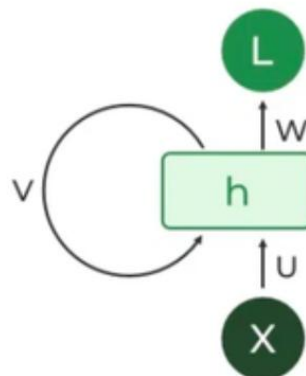
RNNs allow the network to “remember” past information by feeding the output from one step into next step. This helps the network understand the context of what has already happened and make better predictions based on that. For

example when predicting the next word in a sentence the RNN uses the previous words to help decide what word is most likely to come next.



Recurrent Neural Network

This image showcases the basic architecture of RNN and the **feedback loop mechanism where the output is passed back as input for the next time step**. Recurrent Neural Networks (RNNs) solve this **by incorporating loops that allow information from previous steps to be fed back into the network**. This feedback enables RNNs to remember prior inputs making them ideal for tasks where context is important.



RNNs share similarities in input and output structures with other deep learning architectures but differ significantly in how information flows from input to output. Unlike traditional deep neural networks, where each dense layer has distinct weight matrices, RNNs use shared weights across time steps, allowing them to remember information over sequences.

At each time step RNNs process units with a fixed activation function. These units have an internal hidden state that acts as memory that retains information from previous time steps. This memory allows the network to store past knowledge and adapt based on new inputs.

RNN Applications

Recurrent neural networks (RNNs) shine in tasks involving sequential data, where order and context are crucial. Let's explore some real-world use cases. Using RNN models and sequence datasets, you may tackle a variety of problems, including :

Speech Recognition: RNNs power virtual assistants like Siri and Alexa, allowing them to understand spoken language and respond accordingly.

Machine Translation: RNNs translate languages more accurately, like Google Translate by analysing sentence structure and context.

Text Generation: RNNs are behind chatbots that can hold conversations and even creative writing tools that generate different text formats.

Time Series Forecasting: RNNs analyze financial data to predict stock prices or weather patterns based on historical trends.

Music Generation: RNNs can generate music by learning patterns from existing pieces and generating new melodies or accompaniments.

Video Captioning: RNNs analyze video content and automatically generate captions, making video browsing more accessible.

Anomaly Detection: RNNs can learn normal patterns in data streams (e.g., network traffic) and detect anomalies that might indicate fraud or system failures.

Sentiment Analysis: RNNs can analyze sentiment in social media posts, reviews, or surveys by understanding the context and flow of text.

Stock Market Recommendation: RNNs can analyze market trends and news to suggest potential investment opportunities.

Sequence study of the genome and DNA: RNNs can analyze sequential data in genomes and DNA to identify patterns and predict gene function or disease risk.

```
#building the model
model = keras.Sequential([
    layers.LSTM(128, input_shape=(timesteps, features)), # No Embedding for image-like data
    layers.Dense(num_classes, activation='softmax') # Softmax for classification
])
```